Computing Argumentation in Polynomial Number of BDD Operations: A Preliminary Report

Yuqing Tang[1], Timothy J. Norman[2], and Simon Parsons[1,3]

[1] Dept. of Computer Science, Graduate Center, City University of New York
365 Fifth Avenue, New York, NY 10016, USA
ytang@gc.cuny.edu
[2] Dept of Computing Science, The University of Aberdeen
Aberdeen, AB24 3UE, UK
t.j.norman@abdn.ac.uk
[3] Dept of Computer & Information Science, Brooklyn College, City University of New York,
2900 Bedford Avenue, Brooklyn, NY 11210 USA
parsons@sci.brooklyn.cuny.edu

Abstract. Many advances in argumentation theory have been made, but the exponential complexity of argumentation-based reasoning has made it impractical to apply argumentation theory. In this paper, we propose a binary decision diagram (BDD) approach to argumentation-based reasoning. In the approach, sets of arguments and defeats are encoded into BDDs so that an argumentation process can work on a set of arguments and defeats simultaneously in one BDD operation. As a result, the argumentation can be computed in polynomial number of BDD operations on the number of input sentences.

1 Introduction

Argumentation provides an elegant approach to nonmonotonic reasoning [15] and decision making [17, 26], and now sees wide use as a mechanism for supporting dialogue in multiagent systems [32, 33]. As an approach that has its roots in logic — in many systems of argument, the arguments are constructed using some form of logical inference — the efficiency of reasoning using argumentation is a topic of considerable interest [13, 16, 25] with a number of negative results that stress the fact that generating arguments and establishing properties of arguments can be very costly in computational terms.

In this paper we take a rather different look at the computation of arguments. We have been investigating the creation of multiagent plans [36–38], especially the construction of plans that take into account the communication between agents [34, 35]. In doing so, we have been using a representation, that of quantified boolean formulae (QBFs) and binary decision diagrams (BDDs), which has been widely adopted in symbolic planning in non-deterministic domains. It turns out that this representation provides a way to compute arguments, and given the computational efficiency of planning based on QBFs and BDDs, it seems that it can provide an efficient way to compute arguments.We investigate exactly how efficient this approach is in this paper and conclude that we can carry out many of the basic operations needed to compute arguments in a polynomial number of operations.

Note that we are not claiming to be performing general logical inference in polynomical time. As we explain in detail later in the paper, the "polynomial number of opertions" are operations on the BDD representation, and while this representation in many cases can be constructed compactly from a set of logical formulae, there are some cases in which the size of this representation is exponential in the number of formulae.

2 Background

This section gives the technical background needed by the remainder of the paper, a description of *quantified boolean formulae*, and *binary decision diagrams*.

2.1 Qantified boolean formulae

A propositional language $\mathcal{L}$ based on a set of proposition symbols $\mathcal{P}$ with quantification can be defined by allowing standard connectives $\wedge, \vee, \rightarrow, \neg$ and quantifiers $\exists, \forall$ over the proposition variables. The resulting language is a logic of quantified boolean formulae (QBF) [5]. A *symbol renaming operation*, which we use below, can be defined on $\mathcal{L}$, denoted by $\mathcal{L}[\mathcal{P}/\mathcal{P}']$, which means that a new language is obtained by substituting the symbols of $\mathcal{P}$ with the symbols of $\mathcal{P}'$ where $\mathcal{P}'$ contains the same set of propositions as that of $\mathcal{P}$ but using different symbol names (notice that $|\mathcal{P}'| = |\mathcal{P}|$). Similarly for a formula $\xi \in \mathcal{L}$, if $\boldsymbol{x}$ is a vector of propositional variables for $\mathcal{P}$, then a variable renaming operation can be defined by $\xi[\boldsymbol{x}/\boldsymbol{x}']$ which means that all the appearances of variables $\boldsymbol{x} = x_1 x_2 \ldots x_n$ are substituted by $\boldsymbol{x}' = x'_1 x'_2 \ldots x'_n$ which is a vector of the corresponding variables or constants in $\mathcal{P}'$. In QBF, propositional variables can be universally and existentially quantified: if $\phi[\boldsymbol{x}]$ is a QBF formula with propositional variable vector $\boldsymbol{x}$ and x_i is one of its variables, the existential quantification of x_i in ϕ is defined as $\exists x_i \phi[\boldsymbol{x}] = \phi[\boldsymbol{x}][x_i/FALSE] \vee \phi[\boldsymbol{x}][x_i/TRUE]$ and the universal quantification of x_i in ϕ is defined as $\forall x_i \phi[\mathbf{x}] = \phi[\boldsymbol{x}][x_i/FALSE] \wedge \phi[\boldsymbol{x}][x_i/TRUE]$. Here $FALSE$ and $TRUE$ are two propositional constants representing "true" and "false" in the logic. Quantifications over a set $X = \{x_1, x_2, \ldots, x_n\}$ of variables is defined as sequantial quantifications over each variables x_i in the set:

$$Q_X \xi = Q_{x_n} Q_{x_{n-1}} \ldots Q_{x_1} \xi$$

where Q is either $\exists$ or $\forall$. The introduction of quantification doesn't increase the expressive power of propositional logic but allows us to write concise expressions whose quantification-free versions have exponential sizes [11].

With above language, we can encode sets and relations to manipulate sets of arguments and defeats. Let x be an element of a set $X = 2^{\mathcal{P}}$, x can then be explicitly encoded by a conjunction composed of all proposition symbols in $\mathcal{P}$ in either positive or negative form

$$\xi(x) = \bigwedge_{p_i \in x} p_i \wedge \bigwedge_{p_j \notin x \text{ and } p_j \in \mathcal{P}} \neg p_j$$

where $p_i \in x$ means that the corresponding bit p_i is set to be $TRUE$ in the encoding of x, and $p_j \notin x$ means that the corresponding bit p_j is set to be $FALSE$ in the encoding

Set operator	QBF operator
$X_1 \cap X_2$	$\xi(X_1) \wedge \xi(X_2)$
$X_1 \cup X_2$	$\xi(X_1) \vee \xi(X_2)$
$X_1 \setminus X_2$	$\xi(X_1) \wedge \neg\xi(X_2)$
$x \in X$	$\xi(x) \rightarrow \xi(X)$
$X_1 \subseteq X_2$	$\xi(X_1) \rightarrow \xi(X_2)$

Table 1. The mapping between set operators and QBF operators

of x. We denote that a formula γ can be satisfied in an element x by $x \models \gamma$. Then a set of elements can be characterized by a formula $\gamma \in \mathcal{L}$, with the set denoted by $X(\gamma)$, where $X(\gamma) = \{x | x \models \gamma\}$.[4] Two special sets, the empty set $\emptyset$ and the universal set $\mathcal{U}$, are represented by $FALSE$ and $TRUE$ respectively.

With these notions we can have a mapping between the set operations on states and the boolean operations on formulae as shown in Table 1 when X_1 and X_2 are interpreted as two sets of states.

2.2 Binary decision diagrams

In the above, we have showed the natural connections between the set paradigm and its implicit representation using QBF formulae. Now we will briefly survey that the QBF formulae and the operations over them can be represented and efficiently computed using a data structure called Binary Decision Diagrams (BDD) [5]. In this way, the time and space complexity for exploring the space of arguments and defeats for acceptable arguments can be significantly reduced due to the compact representation provided by BDDs in comparison to explicit search techniques.

A BDD is a rooted directed acyclic graph. The terminal nodes are either $TRUE$ or $FALSE$. Each non-terminal node is associated with a boolean variable x_i, and two BDDs, called left and right, corresponding to the values of the sub-formula when x_i is assign $FALSE$ and $TRUE$ respectively. The value of a QBF formula can be determined by traversing the graph from the root to the leaves following the boolean assignment given to the variables of the QBF formula. The advantage of using BDDs to represent QBF formulae is that most basic operations on QBFs can be performed in linear or quadratic time in terms of the number of nodes used in a BDD representation of the formulae if a special form of BDD, called Reduced Ordered Binary Decision Diagram (ROBDD) [5], is used. A ROBDD is a compact BDD which uses a fixed ordering over the variables from the root to the leaves in the BDD, merges duplicate subgraphs into one, and directs all their incoming edges into the merged subgraph. Following the notation traditionally used in symbolic model checking and AI planning, we will refer to an ROBDD simply as a BDD.

Let ξ, ξ_1, ξ_2 be QBF formulae, let the number of nodes used in its BDD representation denoted by $||\cdot||$. With this BDD representation, the complexity of a QBF binary operator $\langle op \rangle$ (e.g. $\wedge, \vee, \rightarrow$) on two formulae ξ_1 and ξ_2, namely $\xi_1 \langle op \rangle \xi_2$, is

[4] Note that $X(p_1 \wedge p_2 \wedge \ldots \wedge p_k) \neq \{s\}$ where $x = \{p_1, p_2, \ldots, p_k\}$.

QBF/Set operator	BDD operator	Complexity
$\neg\xi$	$\neg G(\xi)$	$O(\|\|\xi\|\|)$
$\exists x_i(\xi)$	$G(\xi_{x_i=0}) \vee G(\xi_{x_i=1})$	$O(\|\|\xi\|\|^2)$
$\forall x_i(\xi)$	$G(\xi_{x_i=0}) \wedge G(\xi_{x_i=1})$	$O(\|\|\xi\|\|^2)$
$\xi_1 \wedge \xi_2$	$G(\xi_1) \wedge G(\xi_2)$	$O(\|\|\xi_1\|\| \cdot \|\|\xi_2\|\|)$
$\xi_1 \vee \xi_2$	$G(\xi_1) \vee G(\xi_2)$	$O(\|\|\xi_1\|\| \cdot \|\|\xi_2\|\|)$
$\xi_1 \rightarrow \xi_2$	$G(\xi_1) \rightarrow G(\xi_2)$	$O(\|\|\xi_1\|\| \cdot \|\|\xi_2\|\|)$
$\|X\|$	$Sat\text{-}count(G(\xi(X)))$	$O(\|\|\xi(X)\|\|)$

Table 2. The mapping between QBF operators and BDD operators. ξ, ξ_1, ξ_2 are formulae in QBF; $G(\xi), G(\xi_1), G(\xi_2)$ are BDD representations for these formulae.

$O(||\xi_1|| \times ||\xi_2||)$, that of negation $\neg\xi$ is $O(||\xi||)$ (or $O(1)$ if complement edges are introduced to the BDDs), and that of quantification $Q_{x_i}\,(f[\boldsymbol{x}])$, where Q is either $\exists$ or $\forall$, is $O(||f||^2)$ [5, 11] as summarized in Table 2.

The key achievement of using BDDs (and the front end language of QBFs) to represent sets and relations is that the complexity of the operations will depend on the complexity of the BDD representation instead of the size of the sets and relations, and the complexity of the BDD representation of the sets and relations doesn't depend on the size of those sets and relations. Instead, the operations on BDDs are polynomial in the size of the BDD, and so operations on sets and relations will be polynomial in the size of their BDD representation rather than exponential in their size.

3 Set-theoretic argumentation

Having introduced the ideas from QBFs and BDDs, in this section we give an overview of the argumentation system we will capture using them. The framework we use is mostly drawn from the work of Amgoud and her colleagues [1, 2] with some slight modifications. This framework will abstract away the inference procedure by which the arguments are created and only keep track of the premises the arguments are based on. In the next section, we will introduce the inference procedure back into the representation of arguments.

Definition 1. *An argument based on $\Sigma \subseteq \mathcal{L}$ is pair (H, h) where $H \subseteq \Sigma$ and $H \neq \emptyset$ such that*

1. *H is consistent with respect to $\mathcal{L}$,*
2. *$H \vdash h$,*
3. *H is minimal (for set inclusion).*

H is called the support and h is called the conclusion of the argument. $\mathcal{A}(\Sigma)$ denotes the set of all arguments which can be constructed from Σ.

This definition of argument can be understood as a set of constraints on how information can be clustered as arguments. Condition (1) is to ensure that an argument is coherent.

The coherence of an agent's information is defined in terms of the consistency of the language $\mathcal{L}$ in which the information is written. Condition (2) can be understood as insisting that the conclusion of an argument should be supported by a set of information in the sense of inference in the language $\mathcal{L}$. Condition (3) can be understood as saying that no redundant information should appear in an argument.

Definition 2. *(H', h') is a subargument of the argument (H, h) iff $H' \subseteq H$.*

Definition 3. *Let (H_1, h_1), (H_2, h_2) be two arguments of $\mathcal{A}(\Sigma)$.*

1. *(H_1, h_1) rebuts (H_2, h_2) iff $h_1 \equiv \neg h_2$.*
2. *(H_1, h_1) undercuts (H_2, h_2) iff $\exists h \in H_2$ such that $h_1 \equiv \neg h$.*
3. *(H_1, h_1) contradicts (H_2, h_2) iff (H_1, h_1) rebuts a subargument of (H_2, h_2).*

The binary relations rebut, undercut, *and* contradict *gather all pairs of arguments satisfying conditions (1), (2) and (3) respectively.*

Definitions of rebut, undercut, and contradict will be given below and we will collectively refer to the relations as defeat if no distinction is necessary. Following Dung's work [15], we have the following component definitions:

Definition 4. *An* argumentation framework *is a pair, $Args = \langle \mathcal{A}, \mathcal{R} \rangle$, where $\mathcal{A}$ is a set of arguments, and $\mathcal{R}$ is the binary relation* defeat *over the arguments.*

Definition 5. *Let $\langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework, and $S \subseteq \mathcal{A}$. An argument A is defended by S iff $\forall B \in \mathcal{A}$ if $(B, A) \in \mathcal{R}$ then $\exists C \in S$ such that $(C, B) \in \mathcal{R}$.*

Definition 6. *$S \subseteq \mathcal{A}$. $\mathcal{F}_{\mathcal{R}}(S) = \{A \in \mathcal{A} | A$ is defended by S with respect to $\mathcal{R}\}$.*

Now, for a function $F : D \rightarrow D$ where D is the domain and the range of the function, a fixed point of F is an $x \in D$ such that $x = F(x)$. When the D is associated with an ordering P — for example, P can be set inclusion over the power set D of arguments — x is a *least fixpoint* of F if x is a least element of D with respect to P and x is a fixed point.

Definition 7. *Let $\langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework. The set of acceptable arguments, denoted by $Acc^F_{\mathcal{R}}$, is the least fixpoint of the function $\mathcal{F}_{\mathcal{R}}$ with respect to set inclusion.*

The least fixpoint semantics can be viewed as a mathematical translation of the principle such that an argument survives if it can defend itself and be defended by a set of arguments which can also survive all the attacks made upon them.

4 Representing arguments in QBFs and BDDs

We now turn our attention to using QBFs and BDDs to represent the components of an argumentation system, and then to perform the computations we need to carry out on that representation.

We can label each item $f_i \in \Sigma$ with a proposition l_i. Namely, we will extend the language $\mathcal{L}$ to contain both the information base Σ and the labels for these sentences. Formally, the proposition symbols can be extended to be $\mathcal{P} = \mathcal{P}_D \cup \mathcal{P}_L$ where $\mathcal{P}_D$ is the set of proposition symbols for the domain information, and $\mathcal{P}_L$ is the set of system proposition symbols labeling the sentences in Σ. Given a finite information base $\Sigma \subseteq \mathcal{L}$, $|\mathcal{P}_L(\Sigma)| = |\Sigma|$, namely each sentence $f_i \in \Sigma$ has a corresponding label l_i.

For any formula ξ in $\mathcal{L}$ based on $\mathcal{P} = \mathcal{P}_D \cup \mathcal{P}_L$, $\xi_D = \exists_{\mathcal{P}_L}\xi$ is the formula with only domain symbols left, and $\xi_L = \exists_{\mathcal{P}_D}\xi$ is the formula with only the label symbols left.

4.1 Labeling

For representational convenience, we define

$$SEL(l_i) = l_i \wedge \bigwedge_{j \neq i} \neg l_j.$$

A sentence f_i of Σ corresponds to a pair $\langle SEL(l_i), f_i \rangle$ which can be represented by $SEL(l_i) \wedge f_i$. Given a set of input information $\Sigma = \{f_i\}$ for $f_i \in \mathcal{L}_D$, a labeling table $\Lambda(\Sigma)$ can be expressed as follows

$$\Lambda(\Sigma) = \{\langle SEL(l_i), f_i \rangle\}$$

where $f_i \in \Sigma$ and $l_i \in \mathcal{P}_L$, and the corresponding QBF representation

$$\xi(\Lambda(\Sigma)) = \bigvee_{f_i \in \Sigma} [SEL(l_i) \wedge f_i]$$

The above $\Lambda(\Sigma)$ expression requires $2 \times |\Sigma|$ QBF/BDD operations. [5] Given a subset $\sigma \subseteq \Sigma$,

$$SEL(\sigma) = \bigwedge_{f_i \in \sigma} (l_i) \wedge \bigwedge_{f_j \notin \sigma} \neg l_j$$

4.2 Consistent subsets

Since the support of an argument is a consistent set of propositions, a natural place to start thinking about argument computation is with the computation of consistent subsets. The set of all consistent subsets of Σ is

$$CONS(\Sigma) = \bigvee_{\sigma \subseteq \Sigma} [SEL(\sigma) \wedge \bigwedge_{f_i \in \sigma} f_i] \quad (1)$$

[5] The first condition of using QBF/BDD is to guarantee a way to express the information/specification that we need with only polynomial, linear, or even logarithmic number of QBF/BDD operations; the second condition is to guarantee that the size of the initial, intermediate, and final BDDs corresponding to the information/specification is small enough to fit into memory. For the second condition, if the size of the BDD explodes we may partition the expression into conjunctions or disjunctions, and modify the algorithms manipulating these BDDs correspondingly to try to avoid the explosion. If this still fails, then it means that the problem cannot be efficiently handled by BDDs. In this case, it usually also means that some aspect of the information required to solve the problem is simply too complex.

Computing the above expression directly requires an exponential number of QBF/BDD operations, so we want to find another way to compute it.

Proposition 1. *$CONS(\Sigma)$ can be constructed using $2 \times |\Sigma| - 1$ operations as follows*

$$CONS(\Sigma) = \bigwedge_{f_i \in \Sigma} [l_i \rightarrow f_i]. \tag{2}$$

Proof. The form of formula 2 follows from

$$\begin{aligned} CONS(\Sigma) &= \bigwedge_{f_i \in \Sigma} [l_i \rightarrow f_i] \\ &= \bigwedge_{f_i \in \Sigma} [l_i \rightarrow (l_i \wedge f_i)] \\ &= \bigwedge_{f_i \in \Sigma} [\neg l_i \vee (l_i \wedge f_i)] \\ &= \bigvee_{\sigma \subseteq \Sigma} [\bigwedge_{f_j \notin \sigma} \neg l_j \wedge \bigwedge_{f_i \in \sigma} (l_i \wedge f_i)] \\ &= \bigvee_{\sigma \subseteq \Sigma} [SEL(\sigma) \wedge \bigwedge_{f_i \in \sigma} f_i] \end{aligned}$$

$(l_i \rightarrow f_i) \leftrightarrow (l_i \rightarrow (l_i \wedge f_i))$ follows from:

$$\begin{aligned} A \rightarrow B &\leftrightarrow \neg A \vee B \\ &\leftrightarrow (\neg A \vee A) \wedge (\neg A \vee B) \\ &\leftrightarrow \neg A \vee (A \wedge B) \\ &\leftrightarrow A \rightarrow (A \wedge B) \end{aligned}$$

□

With the above expression, we can exclude empty consistent subsets by

$$CONS^+(\Sigma) = CONS(\Sigma) \wedge (\bigvee_{f_i \in \Sigma} l_k)$$

Because we are only interested in non-empty consistent subsets, from here on we will mean $CONS^+(\Sigma)$ when we use $CONS(\Sigma)$. The set of subsets of selected sentences is

$$\begin{aligned} CONS(\Sigma)_L = \; & \exists_{\mathcal{P}_D} \left(\bigvee_{\sigma \in \Sigma} (SEL(\sigma) \wedge \bigwedge_{f_i \in \sigma} f_i) \right) \\ = \; & \bigvee_{\sigma \in \Sigma} (SEL(\sigma) \wedge \left[\exists_{\mathcal{P}_D} \left(\bigwedge_{f_i \in \sigma} f_i \right) \right] \\ = \; & \bigvee_{\sigma \subseteq \Sigma} SEL(\sigma) \end{aligned}$$

As we see, the complexity of $CONS_L(\Sigma)$ is $O(2 \times |\Sigma| - 1 + |\mathcal{P}_D|)$. Let

$$CONJ(\sigma) = SEL(\sigma) \wedge \bigwedge_{f_i \in \sigma} f_i.$$

Proposition 2. *Given a sentence set selector $\sigma \subseteq \Sigma$ represented by $SEL(\sigma)$, if the conjunction of the selected sentences in σ is consistent then it can be expressed as follows*

$$\begin{aligned} CONJ(\sigma) = &\quad SEL(\sigma) \wedge CONS(\Sigma) \\ &= \bigvee_{\sigma \subseteq \Sigma} [SEL(\sigma) \wedge \bigwedge_{f_i \in \sigma} f_i] \end{aligned}$$

Proof. $CONS(\Sigma)$ is a disjunction of conjunctions of all consistent subsets of Σ. Among these conjunctions, $SEL(\sigma)$ only can make the one corresponding to the σ selection true, which is $SEL(\sigma) \wedge \bigwedge_{f_i \in \sigma} f_i$, and others false. Namely $SEL(\sigma) \wedge CONS(\Sigma) = SEL(\sigma) \wedge \bigwedge_{f_i \in \sigma} f_i$ □

Similarly, the set of conjunctions of a set of selected sentences can be expressed by:

$$CONJ(\{\sigma_i\}) = \bigvee_{\sigma_i} [SEL(\sigma_i)] \wedge CONS(\Sigma)$$

With this expression, we will be able to filter conbinations of consistent and inconsistent sets of sentences into consistent sets.

4.3 QBF/BDD representation of arguments

We can extend the language $\mathcal{P}$ further to contain $\mathcal{P} = \mathcal{P}_L \cup \mathcal{P}_D \cup \mathcal{P}_{L,C} \cup \mathcal{P}_{D,C}$ where $\mathcal{P}_{D,C}$ is a set of renaming symbols of $\mathcal{P}_D$ to represent the conclusions of arguments; $\mathcal{P}_{L,C}$ is an optional set of symbols to label an interesting sub-space of conclusions (the ones we want to compute arguments for). For example, if the sentences in Σ and their negations are of interest, then $P_{L,C} = 2log|\Sigma|$ (we don't need to label a set of sentences, instead we just need to label individual setences and their negations so that we need $2log|\Sigma|$ symbols). Similarly, we will denote $\mathcal{P}_D$ by $\mathcal{P}_{D,P}$ for premises when a distinction is needed.

An argument (H, h) in $\mathcal{L}_D$ can then be represented by formula $\xi(H, h)$ in $\mathcal{L}$

$$\xi(H, h) = SEL(H) \wedge \bigwedge_{f_i \in H} f_i \wedge h[\mathcal{P}_{D,C}]$$

where $h[\mathcal{P}_{D,C}]$ means the expression h is in terms of the symbols of $\mathcal{P}_{D,C}$.

The set of all arguments that can be constructed from Σ will be equivalent to

$$\mathcal{A}(\Sigma) = CONS(\Sigma)$$

for the moment by abstracting away the conclusions. Later we will reintroduce the conclusions to the representation during the query for conclusions and the defeat process.

4.4 Arguments for conclusions

We can construct the set of arguments for a set of conclusions all at once as follows. Let us assume that, besides the input information base Σ, we also have a set of conclusions C that we wish to support.

$$C = \{h_k\}$$

with $K = log|C|$ and a set of labeling symbols

$$\mathcal{P}_{L,C} = \boldsymbol{l}_C = \{l_{1,C}, \ldots, l_{K,C}\}$$

and let c_k be defined as $\boldsymbol{l}_C = k$, namely c_k is the encoding of integer k using the boolean symbols of $\mathcal{P}_{L,C}$.

The set of arguments for C based on Σ can be represented as

$$Args(\Sigma, C)_L = \forall_{\boldsymbol{x} \in \mathcal{P}_D \cup \mathcal{P}_{D,C}} \bigvee_{h_k \in C} \bigvee_{\sigma \subseteq \Sigma} [(\bigwedge_{f_i \in \Sigma} f_i \rightarrow h_k) \wedge SEL(\sigma) \wedge c_k)]$$

and results in

$$Args(\Sigma, C) = Args(\Sigma, C)_L \wedge CONS(\Sigma) \wedge \bigvee_{h_k \in C} (c_k \wedge h_k)$$

Proposition 3. *$Args(\Sigma, C)_L$ can be expressed as*

$$\forall_{\boldsymbol{x} \in \mathcal{P}_D \cup \mathcal{P}_{D,C}} CONS(\Sigma)_L \wedge \left[\bigvee_{h_k \in C} (c_k)\right] \wedge \left[\bigvee_{f_i \in \Sigma} (l_i \wedge \neg f_i) \vee \bigvee_{h_k \in C} (c_k \wedge h_k)\right]$$

using $O(2 \times |C|) + O(|\Sigma|) + O(2 \times |\Sigma| + |\mathcal{P}_D|) + O(|\mathcal{P}_D \cup \mathcal{P}_{D,C}|)$ QBF/BDD operations.

Proof. Start with the first two items above,

$$CONS(\Sigma)_L \wedge \left[\bigvee_{h_k \in C} (c_k)\right] \bigvee_{\sigma \subseteq \Sigma} \bigvee_{h_k \in H} [SEL(\sigma) \wedge c_k]$$

Conjoining with the remaining two items $\left[\bigvee_{f_i \in \Sigma} (l_i \wedge \neg f_i) \vee \bigvee_{h_k \in C} (c_k \wedge h_k)\right]$, gives:

$$\begin{aligned}
&\bigvee_{\sigma \subseteq \Sigma} \bigvee_{h_k \in C} \left[SEL(\sigma) \wedge c_k \wedge \left(\bigvee_{f_i \in \Sigma} (l_i \wedge \neg f_i) \vee \bigvee_{h_k \in C} (c_k \wedge h_k)\right)\right] \\
&= \bigvee_{\sigma \subseteq \Sigma} \bigvee_{h_k \in \Sigma} \left[\left(SEL(\sigma) \wedge (\bigvee_{f_i \in \sigma} \neg f_i)\right) \vee (c_k \wedge h_k)\right] \\
&= \bigvee_{\sigma \subseteq \Sigma} \bigvee_{h_k \in \Sigma} \left[SEL(\sigma) \wedge c_k \wedge \left(\bigwedge_{f_i \in \sigma} (f_i) \rightarrow (h_k)\right)\right]
\end{aligned}$$

The first line is derived using $\bigvee_i A_i \wedge \bigvee_j B_j = \bigvee_i \left[A_i \wedge \left(\bigvee_j B_j\right)\right]$. The second line is derived using

$$SEL(\sigma) \wedge (\bigvee_{f_i \in \Sigma} (l_i \wedge \neg f_i)) = SEL(\sigma) \wedge (\bigvee_{f_i \in \sigma} \neg f_i)$$

since:

$$SEL(\sigma) \wedge (l_i \wedge \neg f_i) = FALSE$$

for any $f_i \notin \sigma$. The second line also employs $c_k \wedge \bigvee_{h_k \in C} (c_k \wedge h_k) = c_k \wedge h_k$ □

Algorithm 4.1 Computing BDD for set-inclusion $\subseteq$

1: Associate with each element in $f_i \in \Sigma$ two BDD variables l_i and l'_i.
2: Take the variable order $l_1, l'_1, l_2, l'_2, ..., l_n, l'_n$ $(n = |\Sigma|)$
3: **for** each l_i **do**
4: link $l_i = 1$ to l'_i
5: link $l_i = 0$ to l_{i+1}
6: **end for**
7: **for** each $l'_i \neq l'_n$ **do**
8: link $l'_i = 1$ to l_{i+1}
9: link $l'_i = 0$ to terminal 0
10: **end for**
11: link $l'_n = 0$, to terminal 0
12: link $l'_n = 1$, to terminal 1

4.5 Minimization of consistent sets with respect to conclusions

In the above, $Args(\Sigma, C)$ may contain non-minimal arguments. To overcome this, we need to minimize the arguments in $Args(\Sigma, C)$ with respect to their conclusions. Given a set of arguments $Q \subseteq \mathcal{A}$ and a partial relation $B \subseteq \mathcal{A} \times \mathcal{A}$ (e.g. the set-inclusion $\subseteq$ relation on the supports of arguments) on $\mathcal{A}$, the set of minimal arguments in Q with respect to B is

$$Min(Q, B) = \{A \in Q | \text{for all } C \in Q, (C, A) \in B \text{ implies } (A, C) \in B\}$$

By encoding Q with a QBF formula $Q[\mathcal{P}]$ based on a set $\mathcal{P}$ of propositional symbols, and encoding the partial relation B with another QBF $B[\mathcal{P}, \mathcal{P}']$ with the first component of B based on symbols in $\mathcal{P}$ and the second component of B based on the symbols in $\mathcal{P}'$, we can compute $Min(Q, B)$ as fllows

$$Min(Q, B) = Q \wedge \forall_{\mathcal{Z}}[(Q[\mathcal{P}/\mathcal{Z}] \rightarrow (B[\mathcal{P}/\mathcal{Z}, \mathcal{P}'/\mathcal{P}] \rightarrow B[\mathcal{P}'/\mathcal{Z}])]$$

where $\mathcal{Z}$ is a temporary set of symbols renamed from $\mathcal{P}$ to hold the intermediate results during the computation.

The set-inclusion relation between two sets of supports $H_1[\mathcal{P}]$ and $H_2[\mathcal{P}']$ can be implemented as:

$$\xi(\subseteq) = \bigwedge_{f_i \in \Sigma} [l_i \rightarrow l'_i].$$

This requires $2 \times |\Sigma|$ QBF/BDD operations to construct. A linear BDD size implementation of $\subseteq$ on the supports of $\mathcal{A}$ is given in Algorithm 4.1 [6].

The set of minimal supports which attack a sentence $h_k \in C$ can be computed as

$$Args_{min}(\Sigma, h_k) = Min\left((Args(\Sigma, C) \wedge c_k)_L, \xi(\subseteq)\right).$$

The set of minimal supports with respect to each each setence in C can be computed as

$$Args_{min}(\Sigma, h_k) = \bigvee_{h_k \in C} Args_{min}(\Sigma, h_k).$$

For description convenience, below we will use $Args(\Sigma, C)$ for $Args_{min}(\Sigma, C)$.

[6] To the best of our knowledge only an exponential implementation exists in the literature [3].

4.6 A QBF representation of defeat

A defeat $\mathsf{defeat}((H,h),(H',h'))$ can be represented by

$$\begin{aligned}\xi(H,h,H',h') = CONJ(H) \wedge SEL(h)[\mathcal{P}_{L,C}] \wedge h[\mathcal{P}_{D,C}] \\ \wedge CONJ(H')[\mathcal{P}'_D] \wedge SEL(h')[\mathcal{P}'_{L,C}] \wedge h'[\mathcal{P}'_{D,C}]\end{aligned}$$

by extending the language $\mathcal{L}[\mathcal{P}]$ to be $\mathcal{L}[\mathcal{P}] \cup \mathcal{L}[\mathcal{P}']$. A defeat relation $D = \{(A_i, A'_i)\}$ can be represented by a single QBF/BDD formula:

$$\xi(D) = \bigvee_{(A_i,A'_i)\in D} [\xi(A_i) \wedge \xi(A'_i)].$$

Now we need an expression with a polynomial number of operations to generate the set of all possible defeats from Σ. To do this, we need to inspect the specific types of defeats. We start with undercut:

Definition 8. *An argument* (H_1, h_1) undercuts *another argument* (H_2, h_2) *iff there exists an* $f \in H_2$ *such that* $h_1 \equiv \neg f$.

and this gives us:

Proposition 4. *Let* $C = \Sigma \cup \{\neg f_i | f_i \in \Sigma\}$*, the set of all possible undercuts can be constructed as*

$$\mathsf{undercut}(\Sigma) = Args(\Sigma, C) \wedge Args(\Sigma', C') \wedge (\bigvee_{f'_i\in\Sigma'}(c_{\neg f_i} \wedge l_i))$$

where $c_{\neg f_i}$ *denotes the encoding of the label that corresponds to* $\neg f_i$ *in* C.

Proof. $Args(\Sigma, C)$ and $Args(\Sigma', C')$ constructs the arguments for C based on Σ using two sets of symbols, and the corresponding selection of input sentences and conclusion sentences. $(\bigvee_{f'_i\in\Sigma'}(c_{\neg f_i} \wedge l_i))$ builds up the undercut relation between these two sets of arguments. □

Note that the setting of the conclusion points $C = \Sigma \cup \{\neg f_i | f_i \in \Sigma\}$ can be changed according to any application-dependent argumentation process, for example $CONS(\Sigma)$ and their negations or other application oriented conclusions and their negations.

Next we consider rebut:

Definition 9. (H_1, h_1) rebuts (H_2, h_2) *iff* $h_1 \equiv \neg h_2$.

We can construct the rebut relation in the same way as undercut by assuming a set of interesting conclusion points. However, we can also construct the rebut relation in the following way and leaving the conclusion points open to make the system more flexible.

Definition 10. *Given a set* H *of sentences, let*

$$\begin{aligned}S(H) &= \{s | s \models H\} \\ S(h) &= \{s | s \models h\}\end{aligned}$$

where s *is an assignment to* $\mathcal{P}$. $H \vdash h$ *iff* $S(H) \subseteq S(h)$.

The definition of rebut is then:

Definition 11. *H_1 rebuts H_2, if there is some h such that $H_1 \vdash h$ and $H_2 \vdash \neg h$.*

and we have:

Proposition 5. *Given two consistent sets of sentences H_1 and H_2, H_1 rebuts H_2 iff $S(H_1) \cap S(H_2) = \emptyset$, namely $[CONJ(H_1) \wedge CONJ(H_2)] \leftrightarrow FALSE$.*

Proof. If H_1 rebuts H_2, then there is a h such that $H_1 \vdash h$ and $H_2 \vdash \neg h$. Since $S(h) \cap S(\neg h) = \emptyset$, and $S(H_1) \subseteq S(h)$ and $S(H_2) \subseteq S(\neg h)$, therefore $S(H_1) \cap S(H_2) = \emptyset$.

If $S(H_1) \cap S(H_2) = \emptyset$, the rebutting point h can be constructed as follows. Let $padding = \neg(H_1 \vee H_2)$, and $h = H_1 \vee padding$. In this way, $S(padding) = \mathcal{U} \setminus (S(H_1) \cup S(H_2))$, $S(h) = S(H_1) \cup S(padding)$, $S(\neg h) = \mathcal{U} \setminus (S(H_1) \cup S(padding)) = S(H_2)$. Therefore $S(H_1) \subseteq s(h)$ and $S(H_2) \subseteq S(\neg h)$, namely h is the rebutting point we are looking for such that $H_1 \vdash h$ and $H_2 \vdash \neg h$. □

Actually h can be anything such that $S(H_1) \subseteq S(h) \subseteq (S(H_1) \cup S(padding))$, so we have the following corollary.

Corollary 1. *Given two sets of sentences H_1 and H_2 which rebut each other, the rebut point h can be obtained by setting $S(H_1) \subseteq S(h) \subseteq S(H_1) \cup S(padding)$ where $padding = \neg(H_1 \vee H_2)$. The choice of $h = H_1 \vee \neg(H_1 \vee H_2)$ which makes H_1 and H_2 be the minimal sets of sentences such that $H_1 \vdash h$ and $H_2 \vdash \neg h$* □

As a result, the set of all rebuts can be expressed as

$$\mathsf{rebut}(\Sigma) = \bigvee_{\sigma \subseteq \Sigma, \sigma' \subseteq \Sigma} [CONJ(\sigma) \wedge CONJ'(\sigma') \wedge \neg (CONJ(\sigma)_D \wedge CONJ(\sigma')_D)]$$

and we have:

Proposition 6. $\mathsf{rebut}(\Sigma)$ *can be expressed as*

$$\mathsf{rebut}(\Sigma) = CONS(\Sigma) \wedge CONS'(\Sigma) \wedge \left[\bigvee_{f_i \in \Sigma}(l_i \wedge \neg f_i) \vee \bigvee_{f_j \in \Sigma}(l'_j \wedge \neg f_j)\right]$$

using $2 \times O(CONS(\Sigma)) + 6 \times |\Sigma| + 3$ QBF/BDD operations.

Proof.

$$\begin{aligned}
&\mathsf{rebut}(\Sigma) \\
&= \bigvee_{\sigma \subseteq \Sigma, \sigma' \subseteq \Sigma} [CONJ(\sigma) \wedge CONJ'(\sigma') \wedge \neg (CONJ(\sigma)_D \wedge CONJ(\sigma')_D)] \\
&= \bigvee_{\sigma \subseteq \Sigma, \sigma' \subseteq \Sigma} [CONJ(\sigma) \wedge CONJ'(\sigma') \wedge (\neg CONJ(\sigma)_D \vee \neg CONJ(\sigma')_D)] \\
&= \bigvee_{\sigma \subseteq \Sigma, \sigma' \subseteq \Sigma} \left[CONJ(\sigma) \wedge CONJ'(\sigma') \wedge \left(\bigvee_{f_i \in \sigma} \neg f_i \vee \bigvee_{f_j \in \sigma'} \neg f_j\right)\right]
\end{aligned}$$

$$= \bigvee_{\sigma \subseteq \Sigma, \sigma' \subseteq \Sigma} \left[CONJ(\sigma) \wedge CONJ'(\sigma') \wedge \left(\bigvee_{f_i \in \sigma} (l_i \wedge \neg f_i) \vee \bigvee_{f_j \in \sigma'} (l'_j \wedge \neg f_j) \right) \right]$$

$$= CONS(\Sigma) \wedge CONS'(\Sigma) \wedge \left[\bigvee_{f_i \in \Sigma} (l_i \wedge \neg f_i) \vee \bigvee_{f_j \in \Sigma} (l'_j \wedge \neg f_j) \right]$$

□

Finally we consider the computation of the contradict relation:

Definition 12. (H_1, h_1) *contradicts* (H_2, h_2) *if and only if* (H_1, h_1) *rebuts a subargument of* (H_2, h_2).

The contradict relation can be computed by

$$\mathsf{contradict}(\Sigma) = \exists_{\mathcal{Z}} \left(\mathsf{rebut}(\Sigma)[\mathcal{P}'/\mathcal{Z}] \wedge \xi(\subseteq)[\mathcal{P}/\mathcal{Z}] \right)$$

4.7 Computing fixed points of argumentation

The relations undercut, rebut and contradict give us the relationship between individual arguments, but, as is usual, we are more interested in computing things like which arguments are *acceptable*, where such properties are defined as fixed-points.

Definition 13. *An argument H defends another argument H' if there exists another argument H'' such that H'' defeats H' but H defeats H''.*

The defend relation can be constructed from the defeat relation on the set of arguments as follows:

$$\mathsf{defend}(\Sigma, \mathsf{defeat}) = \exists_{\mathcal{Z}} \left(\mathsf{defeat}(\Sigma)[\mathcal{P}'/\mathcal{Z}] \wedge \mathsf{defeat}(\Sigma)[\mathcal{P}/\mathcal{Z}] \right)$$

where $\mathsf{defeat}(\Sigma)$ is either $\mathsf{undercut}(\Sigma)$, $\mathsf{rebut}(\Sigma)$, $\mathsf{contradict}(\Sigma)$, or any disjunction of the relations (e.g. $\mathsf{undercut}(\Sigma) \vee \mathsf{rebut}(\Sigma)$). The composition of two relations R_1 and R_2 on the set $\mathcal{A}$ of arguments can be computed by

$$ComposeR(R_1, R_2) = \exists_{\mathcal{Z}} R_1[\mathcal{P}'/\mathcal{Z}] \wedge R_2[\mathcal{P}/\mathcal{Z}].$$

With these constructs defined, the fixed point of argumentation can be computed using Algorithm 4.2. In Algorithm 4.2, the closure of a binary relation R on $\mathcal{A}$, is computed using a method called iterative squaring [7] which is guaranteed to terminate within $O(log|\mathcal{A}|)$ steps. In line $3 : OldR \leftarrow I_{\mathcal{P}_L} \cup \mathsf{defend}_{\mathcal{P}_L}$, the defend relation is first projected to sentence labeling symbols so that during the computation of the defending closure only the label of arguments are considered without referring to their internal structure; the union with the identity relation $I_{\mathcal{P}_L} = \bigwedge_{f_i \in \Sigma} (l_i \leftrightarrow l'_i)$ is to keep the defended arguments in the closure.

Proposition 7. *Algorithm 4.2 computes the fixed point of the defend relation, namely the set of acceptable arguments constructed from Σ.*

Algorithm 4.2 Computing Fixed Point of Argumentation

1: **function** $ComputeFixedpoint(\Sigma, \mathsf{defeat})$ {
   (1) Σ: The set of input information
   (2) defeat is binary relation on $\mathcal{A}$ }
2: $\mathsf{defend} \leftarrow \mathsf{defend}(\Sigma, \mathsf{defeat})$
3: $OldR \leftarrow I_{\mathcal{P}_L} \cup \mathsf{defend}_{\mathcal{P}_L}$
4: $R \leftarrow FAIL$
5: **while** $(OldR \neq R)$ **do**
6: $\quad tmpR \leftarrow R$
7: $\quad R \leftarrow ComposeR(OldR, OldR)$
8: $\quad OldR \leftarrow tmpR$
9: **end while**
10: $Undefeated \leftarrow CONS(\Sigma) \wedge \neg\,(\exists_{\boldsymbol{x} \in \mathcal{P}} \mathsf{defeat})\,[\mathcal{P}'/\mathcal{P}]$
11: $Acc \leftarrow \exists_{\boldsymbol{x} \in \mathcal{P}}(Undefeated \wedge R)[\mathcal{P}'/\mathcal{P}] \vee Undefeated$
12: **return** Acc **end function**

Proof. Let $step(R)$ be the maximum length of paths between a pair $(A, A') \in R$ in the induced graph of the defend relation defend. Let the starting R in line 3 denoted by $R_0 = \mathsf{defend} \cup I$. In R_0, for every $(A, A') \in R$, either $(A, A') \in \mathsf{defend}$, namely A defends A' using one step, or A is identical to A' namely A defends A' using 0 step, therefore $step(R_0) = 1$. Let the consequent content of R in each $while$ iteration denoted by R_i where i is the number of the iteration. Each time, when $R_{i+1} \leftarrow ComposeR(R_i, R_i)$ is applied in line 7, R_{i+1} will gather all the argument pairs of the form (A, A') such that A defends A' using defending steps less or equal than $step(R_{i+1}) = 2 \times step(R_i)$ steps. Assume that i is the number such that $R_{i+1} = R_i$, if the iteration continues we will have

$$R_{i+2} = ComposeR(R_{i+1}, R_{i+1}) = ComposeR(R_i, R_i) = R_{i+1} = R_i$$

namely for all $j \geq i$, $R_j = R_i$. Theforefore, after the $while$ loop terminates R will gather all the argument pairs (A, A') via any number of defending steps. Since the number of arguments is finite, all the defending paths are of finite length, therefore the algorithm is guaranteed to terminate. □

Proposition 8. *The complexity of algorithm 4.2 is $O(|\Sigma| \times K^2 \times |\mathcal{P}|)$ where K is the maximum size of the BDDs which appear during the fixed point computing process.*

Proof. As the analyzed in the proof of proposition 7, the $step(R_i) = step^2(R_{i+1})$. The maximum possible step of R_is is the number of arguments which is $2^{|\Sigma|}$. Therefore the algorithm is guaranteed to terminate after $m = log_2 2^{|\Sigma|} = |\Sigma|$ iterations, therefore the number of iteraction is bounded above by $O(|\Sigma|)$. In each iteration, $CompoeseR$ can be computed using $O(1 + |\mathcal{P}|)$ number of BDD operations, each operation is of complexity $O(K^2)$ where K is the maximum size of BDDs used. Therefore the whole algorithm is bounded above by $O(|\Sigma|) \times O(K^2 \times |\mathcal{P}|)$. □

5 Discussion

Proposition 8 shows that we can compute the fixed-point in a polynomial number of BDD operations. As we mentioned above, this is a long way from saying that we can do general logical inference in polynomial time, rather what we are saying is that while the complexity of algorithm 4.2 depends on the maximum size of the BDD (K), this doesn't depend on the size of Σ but rather on the complexity of the information contained in Σ. In the worse case, K can still be exponential in $|\mathcal{P}|$, but in many practical applications K tends to be small.

Because of this feature of systems built using the QBF/BDD representation, there has been a lot of work on reducing the size of BDDs. Many successful approaches have been developed in literature, especially those developed for symbolic model checking in software and hardware verfication [24], and in non-determinstic AI planning [10]. Examples of techniques for reducing the size of BDDs are early quantification [19], quantification scheduling [9], transition partitioning [6], iterative squaring [7, 8], frontier simplification [12], input splitting [28, 29], and state set A^* branching [22, 21, 23] (a BDD version of the A^* search heuristic [31]).

Another factor affecting the BDD size greatly is variable ordering. The problem of finding an optimal variable ordering is NP-complete [4]. Algorithms based on dynamic programming [14], heuristics [20], dynamic variable reordering [30] and machine learning approaches [18] have been proposed for finding a good variable ordering in reasonable time[7].

We are currently working on an implementation of the reasoning mechanism proprosed above with the aim of experimentally clarifying the nature of K for different argumentation problems.

6 Conclusions and Future Work

In this paper, we have proposed a symbolic model checking approach to compute argumentation. The computation only uses a polynomial number of BDD operations in terms of the number of sentences in the input and the number of symbols used in the input. A key idea in the approach is to construct the set of consistent arguments all together using a polynomial number of BDD operations. In the same way, the defeat relation among these arguments can also be computed all at once using a polynomial number of BDD operations. And with the iterative squaring technique, we are able to compute the fixed point of a set of arguments in polynomial number of BDD operations.

We are currently working on implementing the BDD-based argumentation system proposed in this paper, with the aim of conducting experiments to classify the nature of the BDDs constructed for argumentation. This will allow us to determine how effective this approach will be in general. This in turn may lead us to look for new heuristics for controlling the size of the BDDs we need to construct to compute arguments. Another direction that we are working on is to extend the current method to compute more

[7] [18] is also a good source for other references on BDD variable (re-)ordering.

sophisticated and controllable approaches argumentation, such as those based on argumentation schemes [27]. On the way, we will need to develop BDD techniques to efficiently specify application-dependent patterns of arguments (such as those captured by argument schemes), specify application-dependent patterns of defeats (defeat schemes), and extend the basic entailment-based reasoning modelled here to specify the necessary patterns of rule-based procedural reasoning. In combination with our continuing efforts to use BDD techniqes in multiagent planning and dialogues [34, 36, 38, 39], all these efforts are aimed at our ultimate goal of a practical argumentation-based dialogue model for multiagent planning.

Acknowledgment

Research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

1. Leila Amgoud and Claudette Cayrol. Inferring from inconsistency in preference-based argumentation frameworks. *Journal of Automated Reasoning*, 29(2):125–169, 2002.
2. Leila Amgoud and Claudette Cayrol. A reasoning model based on the production of acceptable arguments. *Annals of Mathematics and Artificial Intelligence*, 34(1-3):197–215, 2002.
3. Rudolf Berghammer and Alexander Fronk. Exact computation of minimum feedback vertex sets with relational algebra. *Fundam. Inf.*, 70(4):301–316, 2005.
4. Beate Bollig and Ingo Wegener. Improving the variable ordering of OBDDs is NP-Complete. *IEEE Transations on Computers*, 45(9):993–1002, 1996.
5. Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.
6. J. R. Burch, E. M. Clarke, and D. E. Long. Symbolic model checking with partitioned transition relations. In *Proceedings of International Conference on Very Large Scale Integration*, pages 49–58. North-Holland, 1991.
7. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 1–33, Washington, D.C., 1990. IEEE Computer Society Press.
8. Gianpiero Cabodi, Paolo Camurati, Luciano Lavagno, and Stefano Quer. Disjunctive partitioning and partial iterative squaring: an effective approach for symbolic traversal of large circuits. In *DAC '97: Proceedings of the 34th Annual Conference on Design Automation*, pages 728–733, New York, NY, USA, 1997. ACM.

9. Pankaj Chauhan, Edmund M. Clarke, Somesh Jha, Jim Kukula, Tom Shiple, Helmut Veith, and Dong Wang. Non-linear quantification scheduling in image computation. In *ICCAD '01: Proceedings of the 2001 IEEE/ACM International Conference on Computer-aided Design*, pages 293–298, Piscataway, NJ, USA, 2001. IEEE Press.
10. A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1-2):35–84, 2003.
11. O. Coudert and J. C. Madre. The implicit set paradigm: a new approach to finite state system verification. *Formal Methods in System Design*, 6(2):133–145, 1995.
12. Olivier Coudert, Christian Berthet, and Jean Christophe Madre. Verification of synchronous sequential machines based on symbolic execution. In *Automatic Verification Methods for Finite State Systems*, pages 365–373, 1989.
13. Yannis Dimopoulos, Bernhard Nebel, and Francesca Toni. On the computational complexity of assumption-based argumentation for default reasoning. *Artificial Intelligence*, 141:57–78, 2002.
14. Rolf Drechsler, Nicole Drechsler, and Wolfgang Günther. Fast exact minimization of BDDs. In *DAC '98: Proceedings of the 35th Annual Conference on Design Automation*, pages 200–205, New York, NY, USA, 1998. ACM.
15. Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–358, 1995.
16. Paul E. Dunne and T. J. M. Bench-capon. Two party immediate response disputes: properties and efficiency. *Artificial Intelligence*, 149:2003, 2001.
17. Thomas Gordon and Nikos Karacapilidis. The zeno argumentation framework. In *in Proceedings of the Sixth International Conference on AI and Law*, pages 10–18. ACM Press, 1997.
18. Orna Grumberg, Shlomi Livne, and Shaul Markovitch. Learning to order BDD variables in verification. *Journal of Artificial Intelligence Research (JAIR)*, 18:83–116, 2003.
19. Ramin Hojati, Sriram C. Krishnan, and Robert K. Brayton. Early quantification and partitioned transition relations. In *ICCD '96: Proceedings of the 1996 International Conference on Computer Design, VLSI in Computers and Processors*, pages 12–19, Washington, DC, USA, 1996. IEEE Computer Society.
20. Jawahar Jain, William Adams, and Masahiro Fujita. Sampling schemes for computing OBDD variable orderings. In *ICCAD '98: Proceedings of the 1998 IEEE/ACM International Conference on Computer-aided Design*, pages 631–638, New York, NY, USA, 1998. ACM.
21. Rune M. Jensen, Randal E. Bryant, and Manuela M. Veloso. An efficient BDD-based A* algorithm. In *Proceedings of AIPS-02 Workshop on Planning via Model Checking*, 2002.
22. Rune M. Jensen, Randal E. Bryant, and Manuela M. Veloso. SetA*: An efficient BDD-based heuristic search algorithm. In *Proceedings of 18th National Conference on Artificial Intelligence (AAAIŠ02)*, pages 668–673, 2002.
23. Rune M. Jensen, Manuela M. Veloso, and Randal E. Bryant. State-set branching: Leveraging BDDs for heuristic search. *Artificial Intelligence*, 172(2-3):103–139, 2008.
24. Ranjit Jhala and Rupak Majumdar. Software model checking. *ACM Comput. Surv.*, 41(4):1–54, 2009.
25. Antonios C. Kakas and Francesca Toni. Computing argumentation in logic programming. *Journal of Logic and Computation*, 9:515–562, 1999.
26. Nikos Karacapilidis and Dimitris Papadias. Computer supported argumentation and collaborative decision making: The hermes system. *Information Systems*, 26:259–277, 2001.
27. Joel Katzav and Chris Reed. On argumentation schemes and the natural classification of arguments. *Argumentation*, 18(2), 2004.

28. Christoph Meinel and Thorsten Theobald. *Algorithms and Data Structures in VLSI Design*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1998.
29. In-Ho Moon, James H. Kukula, Kavita Ravi, and Fabio Somenzi. To split or to conjoin: the question in image computation. In *DAC '00: Proceedings of the 37th conference on Design automation*, pages 23–28, New York, NY, USA, 2000. ACM.
30. Shipra Panda, Fabio Somenzi, and Bernard F. Plessier. Symmetry detection and dynamic variable ordering of decision diagrams. In *ICCAD '94: Proceedings of the 1994 IEEE/ACM International Conference on Computer-aided Design*, pages 628–631, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.
31. Judea Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.
32. Henry Prakken. Coherence and flexibility in dialogue games for argumentation. *Journal of Logic and Computation*, 15(6):1009–1040, 2005.
33. Iyad Rahwan, Sarvapali D. Ramchurn, Nicholas R. Jennings, Peter Mcburney, Simon Parsons, and Liz Sonenberg. Argumentation-based negotiation. *The Knowledge Engineering Review*, 18(4):343–375, December 2003.
34. Yuqing Tang, Timothy J. Norman, and Simon Parsons. A model for integrating dialogue and the execution of joint plans. In *Proceedings of the Eigth International Joint Conference on Autonomous Agents and Multiagent Systems*, Budapest, Hungary, May 10-15 2009.
35. Yuqing Tang, Timothy J. Norman, and Simon Parsons. Towards the implementation of a system for planning team activities. In *Proceedings of the Second Annual Conference of the ITA*, University of Maryland University College, Maryland, 2009.
36. Yuqing Tang and Simon Parsons. Argumentation-based dialogues for deliberation. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 552–559, New York, NY, USA, 2005. ACM Press.
37. Yuqing Tang and Simon Parsons. Using argumentation-based dialogues for distributed plan management. In *Proceedings of the AAAI Spring Symposium on Distributed Plan and Schedule Management*, Stanford, 2006.
38. Yuqing Tang and Simon Parsons. A dialogue mechanism for public argumentation using conversation policies. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 445–452, Estoril, Portugal, May 12-16 2008.
39. Yuqing Tang and Simon Parsons. An MDP model for planning team actions with communication. Technical report, International Technology Alliance in Network and Information Science, 2009.